TUTORIEL PUPPET

tutoriel réalisé par : M. Vidal Maurice et M. Prunac Romain dans le cadre d'un stage pour l'association tetaneutral.net.

Puppet est ce que l'on peut appeler un gestionnaire de configuration.
C'est un outil qui facilite le contrôle et la mise à jour de configurations tout en offrant la possibilité de faire abstraction de l'OS et de l'architecture concernée. Puppet va permettre de déployer des fichiers, des services, des packages, des commandes et même un cron au travers de clients qui deviendront des Serveurs (exemple : BIND, UNBOUND).

Puppet est donc un outil de centralisation de l'administration de systèmes hétérogènes ou homogènes
Au travers du serveur puppet (PuppetMaster), chaque machine, appelée noeud (node), fait tourner Puppetd, qui :
- Applique la configuration initiale pour le noeud concerné.
- Applique les nouveautés de configuration au fil du temps.
- S'assure de manière régulière que la machine correspond bien à la config voulu.

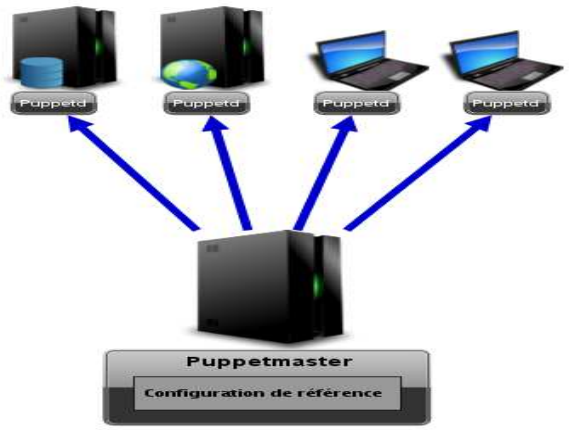
La communication est assurée via des canaux chiffrés, en utilisant le protocole HTTPS de communication de ruby, et donc SSL (une mini-pki est fournie). Puppetmaster sait servir :
- des recettes de configuration (recipes)
- des fichiers
- des modèles (qui sont des fichiers avec des variables de remplacement)
- des objets ruby (c'est là l'extensibilité de l'application)
Et bien évidemment : Puppet est un logiciel libre écrit en Ruby, multiplateforme : BSD (Free, MacOS ...), Linux (Redhat, Debian, Suse ...), Sun (OPENSolaris ...).

Puppet fournit un langage déclaratif simple :
- classes (permet de définir les configurations de chaque service)
- héritage (permet de regrouper les configurations communes)
- types d'objets particuliers (définis par puppet, ou par des modules)
- fonctions utilisateur
- abonnement d'instances à d'autres instances (ex : un service est abonné à son fichier de conf, la modif de l'un entraînera le redémarrage de l'autre)

Exemples de types d'objets internes :

- File, permet de demander qu'un fichier existe, dispose de certains droits, et corresponde éventuellement à un certain contenu, fournit statiquement, ou à travers un modèle (template) avec substitution de variables.
- Service, permet de déclarer un service, de s'assurer qu'il fonctionne, de le redémarrer dans certaines conditions (par exemple quand la configuration est modifiée).
- Package, permet de demander l'installation, la mise à jour ou la désinstallation d'un package, en utilisant le système de packaging natif de l'OS utilisé (ports, deb, packages MacosX etc.)
- User, permet de gérer les utilisateurs UNIX (dans /etc/passwd) indépendemment de l'OS utilisé.

Puppet fonctionne selon le modèle client/serveur :



Pré-requis :

Nous prenons l'exemple que notre serveur Puppet à pour hostname : puppetmaster, et nos clients portent le nom de : puppetcli1, puppetcli2.

Comme nom de domaine nous prenons l'exemple de l'université de Perpignan : univ-perp.fr

Et les adresses IP sont donc :

HOSTNAME	ADRESSE IP
puppetmaster	192.168.1.15
puppetcli1	192.168.1.16
puppetcli2	192.168.1.17

Installation de Puppet

Serveur : **# apt-get install puppetmaster**
Clients : **# apt-get install puppet**

Ici nous utilisons la version 2.6.2 de Puppet. Lors de l'installation, un utilisateur « puppet » sera créé.

Tout d'abord il faut que le fichier hosts de chaque client référence l'adresse IP, le hostname et le FQDN du serveur. Ainsi il faut rajouter dans /etc/hosts :

192.168.1.15 puppetmaster puppetmaster.univ-perp.fr

De même sur le serveur il faut rajouter les noms des 3 machines clientes de la même façon.
A ce stade nous n'avons pas de serveurs DNS, justement nous allons le(s) déployer :)

Il est important (si l'on a un pare-feu) d'ouvrir les ports 8140 coté serveur :
 # iptables -A INPUT -p tcp -m state --state NEW -s 192.168.1.0/24 --dport 8140 -j ACCEPT

De plus il est important de pouvoir regarder les logs (tant sur le serveur que sur les clients) :
 # tailf /var/log/daemon.log

Configuration de Puppet

DEMANDE DE CERTIFICAT

Serveur :
Il faut modifier le fichier /etc/puppet/puppet.conf, il doit ressembler à ceci :

```
[main]
logdir=/var/log/puppet
vardir=/var/lib/puppet
ssldir=/var/lib/puppet/ssl
rundir=/var/run/puppet
factpath=$vardir/lib/facter
templatedir=$confdir/templates

[master]
report=true
reports=log,store
modulepath=/etc/puppet/modules
```

Clients :
Ici il faut rajouter le FQDN (ou Hostname s'il n'y a pas de domaine) du serveur, ainsi dans /etc/puppet/puppet.conf :

```
[main]
...
server=puppetmaster.univ-perp.fr
```

Une fois cette modification faite, il n'y pas plus qu'à demander au serveur Puppet de signer notre certificat, car rappelons que PUPPET utilise le SSL, il est donc plus sécurisé qu'un NFS.

puppetcli1# puppetd --waitforcert 60 --test
ou
puppetcli1# puppet agent --waitforcert 60 --test

Le serveur va ensuite visualiser les demandes :
puppetmaster#puppetca --list

Puis il va signer les demandes :
puppetmaster#puppetca --sign puppetcli1
ou
puppetmaster#puppetca --sign --all (pour tous d'un coup)

Le serveur est prêt à l'emploi, et les clients ont un certificat signé.

Configuration de UNBOUND via PUPPET :

Tout d'abord laissez moi vous montrer à quoi va ressembler l'arborescence de votre serveur puppet :

```
root@machine3:/# tree /etc/puppet/
/etc/puppet/
├── auth.conf
├── fileserver.conf
├── manifests
│   ├── cron.pp
│   ├── modules.pp
│   ├── node.pp
│   └── site.pp
├── modules
│   ├── bind
│   │   ├── files
│   │   │   ├── db.domaine.ext
│   │   │   ├── named.conf.local
│   │   │   ├── named.conf.options
│   │   │   └── rev.domaine.ext
│   │   └── manifests
│   │       └── init.pp
│   └── unbound
│       ├── files
│       │   └── unbound.conf
│       └── manifests
│           └── init.pp
├── namespaceauth.conf
├── puppet.conf
└── templates

9 directories, 15 files
```

Afin d'utiliser Puppet un minimum, nous allons lui demander de déployer UNBOUND sur une machine cliente (puppetcli1). Il va falloir configurer le serveur, ainsi /etc/puppet/fileserver.conf :

```
[files]
path /etc/puppet/files
allow *.univ-perp.fr
```

Nous venons d'autoriser seulement les machines provenant du domaine à pouvoir recevoir des fichiers du répertoire /etc/puppet/files/

Il faut également créer le fichier : /etc/puppet/namespaceauth.conf et y rajouter :

```
[puppetrunner]
allow *
```

Le daemon du client va initialement interroger le fichier /etc/puppet/manifests/site.pp du serveur, c'est pourquoi dans ce fichier nous allons importer tous les fichiers .pp (spécialement conçu pour Puppet).
/etc/puppet/manifests/site.pp :

```
import "node"
import "modules.pp"
import "cron"
```

Ensuite pour éviter de déployer UNBOUND sur tous les nodes, nous allons spécifier lesquels vont recevoir la classe UNBOUND que nous créerons (dans le fichier init.pp du module Unbound). Ainsi dans le fichier /etc/puppet/manifests/node.pp :

```
node basenode {
        include crontab
}
node 'puppetcli1.univ-perp.fr' inherits basenode {
        include unbound, resolv
}
node 'puppetcli2.univ-perp.fr' inherits basenode {
        include bind, resolv
}
```

Ici nous créons un faux node (client puppet) nommé basenode en lui incluant la classe "crontab", et les nodes puppetli1 et puppetcli2 héritent de basenode (autrement dit de la classe crontab). Ensuite nous leurs attribuons la classe unbound et resolv pour puppetcli1 puis bind et resolv pour puppetcli2.

Il faut également créer le fichier /etc/puppet/manifests/modules.pp qui va référencer les modules du serveur :

```
import "unbound"
import "bind"
```

On peut appeler un module, le nom d'un répertoire, en effet il est recommandé de créer des modules pour chaque service afin de rendre la configuration plus souple.

Pour finir sur la configuration de base, nous allons mettre en place un cron qui se déploiera sur les clients afin de lancer une synchronisation avec le serveur, ainsi quand le cron arrive à son échéance (15 minutes) il va permettre de checker toutes modifications du client puppet. Donc on va créer le fichier /etc/puppet/manifests/cron.pp :

```
class crontab {
        cron { "synchronisation puppet":
                command => "/usr/sbin/puppetd --test --server=puppetmaster.univ-perp.fr",
                minute => "*/15"
        }
}
```

Passons à la configuration des modules :

Création des modules :
 # mkdir -p /etc/puppet/modules/unbound/{files,manifests,templates}
 # mkdir -p /etc/puppet/modules/bind/{files,manifests,templates}

Le dossier MANIFESTS permet de répertorier le fichier init.pp qui défini le coeur du module.
Le dossier FILES contient les fichiers statiques, ceux qui ne changent pas d'un node à l'autre.
Le dossier TEMPLATES contient les fichiers de configurations variant d'un hôte à l'autre.

Maintenant que les modules sont crées, nous plaçons le fichier de configuration d'Unbound que voici :
/etc/puppet/modules/unbound/files/unbound.conf :

Ici nous faisons la résolution du domaine : univ-perp.fr avec pour serveur DNS : puppetcli1

```
### Simple recursive caching DNS
## /etc/unbound.conf
#
server:
        verbosity: 1
        outgoing-range: 60
        interface: 192.168.1.16
        do-ip4: yes
        do-udp: yes
        do-tcp: yes
        access-control: 192.168.1.0/24 allow            #On autorise le réseau 192.168.1.0/24 à interroger le DNS
 chroot: ""                                     #Pas de chroot par défaut
 logfile: "/var/log/unbound.log"                        #On défini le fichier de log
 use-syslog: no                                 #Log par défaut dans syslog
 hide-identity: yes
 hide-version: yes
 harden-glue: yes                                       #On dit que le serveur fait autorité sur la zone
private-domain: "univ-perp.fr."                                 #On définit le suffix dns du réseau local
 local-zone: "univ-perp.fr." static                                     #on définit qui s'occupe de quoi dans le
domaine
local-data: "univ-perp.fr. IN MX 10 puppetcli2.univ-perp.fr."           #service de messagerie
local-data: "univ-perp.fr. IN NS puppetcli1.univ-perp.fr."              #service DNS
local-data: "puppetmaster.univ-perp.fr. IN A 192.168.1.15"                      #résolution : nom -> adresse
local-data: "puppetcli1.univ-perp.fr. IN A 192.168.1.16"
local-data: "puppetcli2.univ-perp.fr. IN A 192.168.1.17"
local-data-ptr: "192.168.1.15 puppetmaster.univ-perp.fr."               #résolution inverse : adresse -> nom
local-data-ptr: "192.168.1.16 puppetcli1.univ-perp.fr."
local-data-ptr: "192.168.1.17 puppetcli2.univ-perp.fr."
python:
remote-control:
forward-zone:
        name: "."
        forward-addr: 192.168.1.1     # DNS de la Box
```

Passons à la configuration du fichier principal : /etc/puppet/modules/unbound/manifests/init.pp :

```
class unbound {
        package { unbound: ensure => installed }
        file { "/etc/unbound/unbound.conf":
                notify => Service["unbound"],
                mode => 644,
                owner => "root",
                group => "root",
                source => "puppet:///modules/unbound/unbound.conf"
                }
        service { unbound: ensure => running,
                           enable => true
        }       }
class resolv {
        file { "/etc/resolv.conf":
                ensure => present,
                content => "domain univ-perp.fr
search univ-perp.fr
nameserver $ipaddress"
        }
}
```

Pour la classe "unbound", nous venons de lui permettre d'installer le paquet UNBOUND, de lui déployer le fichier de configuration sur /etc/unbound/unbound.conf afin de remplacer l'existant et ensuite nous lui avons forçer le démarrage du daemon UNBOUND. Lors du déploiement d'un fichier, dans le paramètre SOURCE, remarquons qu'il n'est pas nécessaire de référencer le répertoire "files" dans le module, grâce au protocole qui est défini ici : puppet:///
Quant à la classe "resolv" est de type TEMPLATES, c'est à dire qu'elle est dynamique, elle s'adapte à l'hôte grâce à son adresse IP qui est remonté grâce aux facts (outils facter qui est compréhensible par PUPPET). En effet Facter est un outil associé à puppet, chargé d'énumérer les paramètres locaux d'un système :

- Adresse IP
- Hostname
- Distribution et version
- Toutes autres variables locales définies par l'utilisateur.

Il est disponible sur les noeuds utilisant Puppetd, il fournit des variables utilisables dans les templates puppet.

Configuration du module BIND

Voici le fichier de configuration /etc/puppet/modules/bind/manifests/init.pp :

```
class bind {
        package { bind9: ensure => installed }
        file { "/etc/bind/named.conf.local":
                ensure => present,
                mode => 644,
                owner => "root",
                group => "bind",
                source => "puppet:///modules/bind/named.conf.local" }
        file { "/etc/bind/named.conf.options":
                ensure => present,
                mode => 644,
                owner => "root",
                 group => "bind",
                source => "puppet:///modules/bind/named.conf.options" }
        file { "/var/cache/bind/db.univ-perp.fr":
                ensure => present,
                mode => 644,
                 owner => "root",
                group => "bind",
                source => "puppet:///modules/bind/db.univ-perp.fr" }
        file { "/var/cache/bind/rev.univ-perp.fr":
                ensure => present,
                mode => 644,
                owner => "root",
                 group => "bind",
                source => "puppet:///modules/bind/rev.univ-perp.fr" }
        service { bind9: ensure => running,
                enable => true
        }
}
```

Ici nous installons (si BIND n'est pas installé) le paquet, puis lui déployons les fichiers de configurations, puis le forçons à être démarré. Nous lui attribuons les droits 644 (par défaut sur Bind), puis le propriétaire root et le groupe bind.
Pareillement, ici nous remarquons que pour déployer le fichier, il n'est pas nécessaire de référencer le répertoire "files" dans le module, grâce au protocole qui est défini ici : puppet:///

Passons aux fichiers de configurations : /etc/puppet/modules/bind/files/named.conf.local :

```
// La zone primaire univ-perp.fr
zone "univ-perp.fr" {
    type master;
    file "/var/cache/bind/db.univ-perp.fr";
};
// La zone inverse
zone "1.168.192.in-addr.arpa" {
    type master;
    file "/var/cache/bind/rev.univ-perp.fr";
};
```

Puis le fichier : /etc/puppet/modules/bind/files/named.conf.options :

```
options {
        directory "/var/cache/bind";
forwarders {
        192.168.1.1;
 };
        auth-nxdomain no;    # conform to RFC1035
        listen-on-v6 { any; };
};
```

Le fichier de zone primaire UNIV-PERP.FR : /etc/puppet/modules/bind/files/db.univ-perp.fr :

```
;
; BIND data file for univ-perp.fr
;
$TTL    604800
@       IN      SOA     puppetcli2.univ-perp.fr. admin.univ-perp.fr. (
                201205091         ; Serial
                    604800        ; Refresh
                     86400        ; Retry
                   2419200         ; Expire
                    604800 )      ; Negative Cache TTL
@           IN      NS      puppetcli2.univ-perp.fr.
puppetcli2          IN      A       192.168.1.17
puppetcli1          IN      A       192.168.1.16
puppetmaster        IN          A           192.168.1.15
```

Et pour finir le fichier de zone inverse ARPA : /etc/puppet/modules/bind/rev.univ-perp.fr :

```
;
; BIND data file for 192.168.1
;
$TTL    604800
@       IN      SOA     puppetcli2.univ-perp.fr. (
                201209051         ; Serial
                    604800        ; Refresh
                     86400        ; Retry
                   2419200         ; Expire
                    604800 )      ; Negative Cache TTL
@       IN      NS      puppetcli2.univ-perp.fr.
16      IN      PTR     puppetcli1.univ-perp.fr.
15          IN          PTR         puppetmaster.univ-perp.fr.
```

Configuration des templates

Variables prédéfinie:

Ici nous allons vous montrer comment on peut adapter les fichiers de configurations que nous déployons suivant le node. Ainsi chaque node aura un paramètre différent d'un autre node, il s'agit du template. Nous allons prendre le cas du fichier de configuration d'Unbound. Sur le serveur Puppet il faut renommer le fichier /etc/puppet/modules/unbound/files/unbound.conf en /etc/puppet/modules/unbound/files/unbound.conf**.erb**.

En effet lorsqu'on créé un template, il doit être sous l'extension "erb".
En effet nous allons nous servir de la variable "ipaddress" disponible grâce à l'outil facter, ainsi en étant sur le client, il suffit de lancer la commande : **puppetcli1 # facter** pour obtenir les informations des variables de la machine locale.

Nous allons renseigner la variable "ipaddress" grâce aux balises de début: <%= et aux balises de fin: %>
Ainsi :

Puppetmaster # head /etc/puppet/modules/unbound/files/unbound.conf.erb

```
server:
        verbosity: 1
        outgoing-range: 60
        interface: <%=  ipaddress %>
        do-ip4: yes
        do-udp: yes
        do-tcp: yes
        access-control: 192.168.1.0/24 allow
 chroot: ""
 logfile: "/var/log/unbound.log"
```

Création de variables:

Nous allons maintenant créer des variables qui s'utiliseront uniquement sur les nodes définis. Nous poursuivons sur l'exemple d'Unbound qui se déploiera sur le client : puppetcli1.univ-perp.fr. Dans un premier temps il faut créer la variable dans le fichier /etc/puppet/manifests/node.pp :

```
node 'puppetcli1.univ-perp.fr inherits basenode {
        $network = "192.168.1.0"
        include unbound
}
```

Puis l'appliquer sur le fichier /etc/puppet/modules/unbound/files/unbound.conf.erb :
Puppetmaster # head /etc/puppet/modules/unbound/files/unbound.conf.erb

```
server:
        verbosity: 1
        outgoing-range: 60
        interface: <%=  ipaddress %>
        do-ip4: yes
        do-udp: yes
        do-tcp: yes
        access-control: <%= network %>/24 allow
 chroot: ""
 logfile: "/var/log/unbound.log"
```

La mise en place de templates permet de gagner sensiblement du temps sur la configuration des postes, c'est d'ailleurs pour cette utilisation que l'outil facter a été développé. En effet Puppet sert surtout pour les déploiements massifs, le gain de temps et d'énergie est ainsi effectif.

Mise en place d’une itération

Ici il est inutile de mettre en place une itération, mais je vais vous en montrer une. Dans cet exemple on va déployer le fichier /etc/motd qui sera différents suivant l’hôte distant, en effet il référencera un message différent. Ainsi dans /etc/puppet/manifests/site.pp :

```
$choix = $hostname ? {
puppetcli1 => "/etc/motd"
puppetcli2 => "/etc/motd"
}
file { "$choix" :
        ensure => present,
        source => $hostname ?  {
                puppetcli1 => "puppet:///files/motdpuppetcli1"
                puppetcli2 => "puppet:///files/motdpuppetcli2"
        }
}
```

LANCEMENT DE LA SYNCHRONISATION :

Sur le serveur, il faut initier la connexion :
 # puppetmasterd --no-daemonize --debug

Nous lançon le daemon en mode test, c'est à dire non-démon afin de débugger et vérifier que tout se déroule convenablement.

Si on rencontre un problème tel que :
Could not run: Could not create PID file: /var/run/puppet/master.pid

Il faut alors tuer le processus Puppet sur le serveur : **# killall puppet** et relancer la commande :
 # puppetmasterd --no-daemonize –debug

Attention : Cette commande ne doit jamais être interrompu !

Sur les clients on va vérifier le fichier du serveur /etc/puppet/site.pp avec la commande :
 # puppetd --test --server=puppetmaster.univ-perp.fr

A savoir qu'un cron est mis en place afin d'éviter aux clients de renouveller cette commande manuellement, il faut noter que la commande du serveur : "puppetmasterd --no-daemonize -d" ne doit pas être arrêtée.

BIBLIOGRAPHIE

http://books.sysadmins.su/system/files/Pro.Puppet.pdf

http://media.techtarget.com/searchEnterpriseLinux/downloads/Turnbull_ch2.pdf

http://www.k-tux.com/puppet-gestionnaire-de-configuration/3

http://www.be-root.com/2012/02/02/puppet-utilisation-des-templates/

http://www.puppetcookbook.com/posts/exec-a-command-in-a-manifest.html

http://raisin.u-bordeaux.fr/IMG/pdf/Puppet_RAISIN_final-2.pdf

http://doc.ubuntu-fr.org/puppet

http://www.crium.univ-metz.fr/docs/system/puppet.html

http://www.octopuce.fr/Puppet-Administration-systeme-centralisee

http://bitfieldconsulting.com/puppet-tutorial-2

http://www.craigdunn.org/2010/08/part-2-puppet-2-6-1-configure-puppetmaster-and-puppetd/

http://docs.puppetlabs.com/guides/environment.html

http://projects.puppetlabs.com/projects/puppet/wiki/Simplest_Puppet_Install_Pattern

http://chiliproject.tetaneutral.net/projects/tetaneutral/wiki/PUPPET

http://finninday.net/wiki/index.php/Zero_to_puppet_in_one_day

http://madeinsyria.fr/2011/06/howto-puppet-administration-et-industrialisation-de-masse/

http://jeyg.info/mise-en-place-de-puppet-un-gestionnaire-de-configuration-linuxunix/

http://www.deimos.fr/blocnotesinfo/index.php?title=Puppet_:_Solution_de_gestion_de_fichier_de_configuration

http://bitfieldconsulting.com/puppet-tutorial

http://www.unixgarden.com/index.php/gnu-linux-magazine/les-sysadmins-jouent-a-la-poupee